中国科学院大学
University of Chinese Academy of Sciences

# CONTENTS



# 一、实验目的及要求



## ➢ 实验目的

1.理解和掌握循环神经网络概念及在深度学习框架中的实现。

2.掌握使用深度学习框架进行文本生成任务的基本流程：如数据读取、构造网络、训练和预测等。

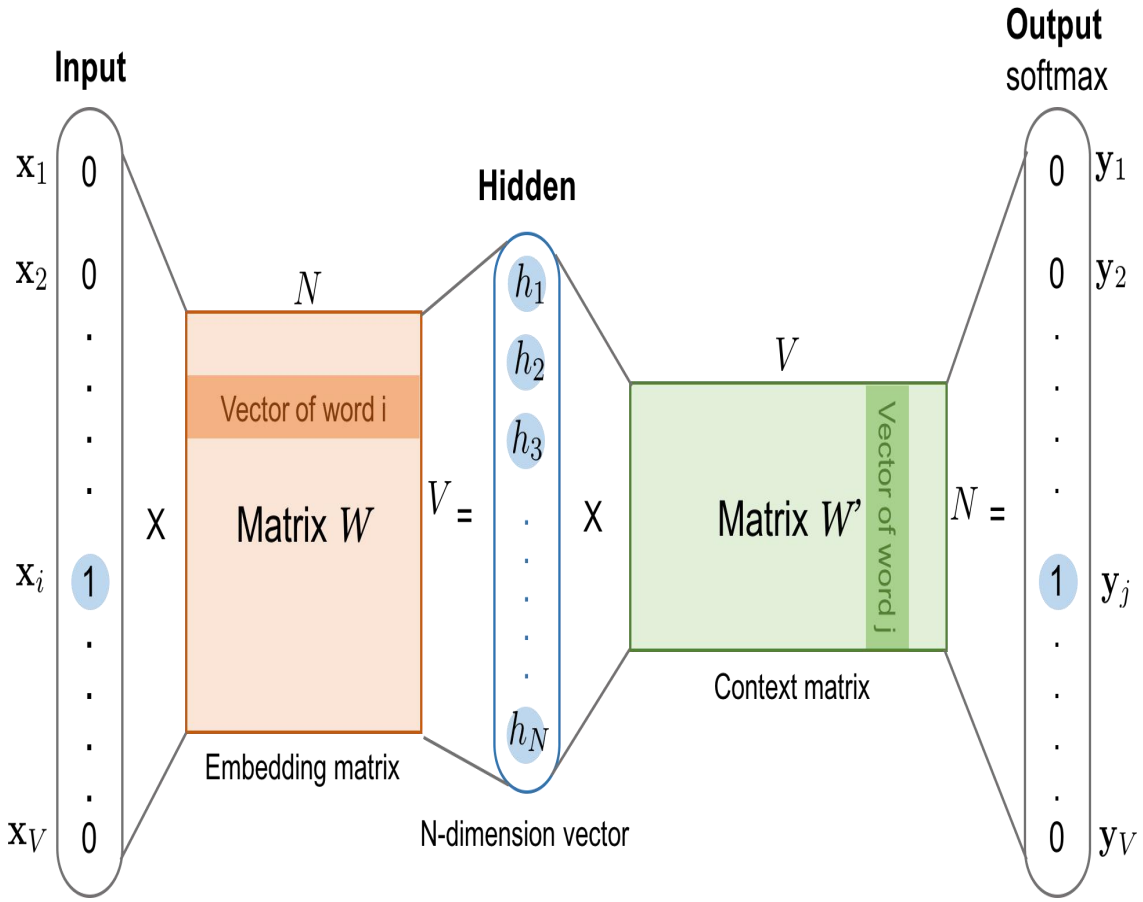
## ➢ 实验要求

1.基于Python语言和任意一种深度学习框架（实验指导书中使用Pytorch框架进行介绍），完成数据读取、网络设计、网络构建、模型训练和模型测试等过程，最终实现一个可以自动写诗的程序。网络结构设计要有自己的方案，不能与实验指导书完全相同。

2.随意给出首句，如给定“湖光秋月两相和”，输出模型续写的诗句。也可以根据自己的兴趣，进一步实现写藏头诗（不做要求）。要求输出的诗句尽可能地满足汉语语法和表达习惯。实验提供预处理后的唐诗数据集，包含57580首唐诗（在课程网站下载），也可以使用其他唐诗数据集。

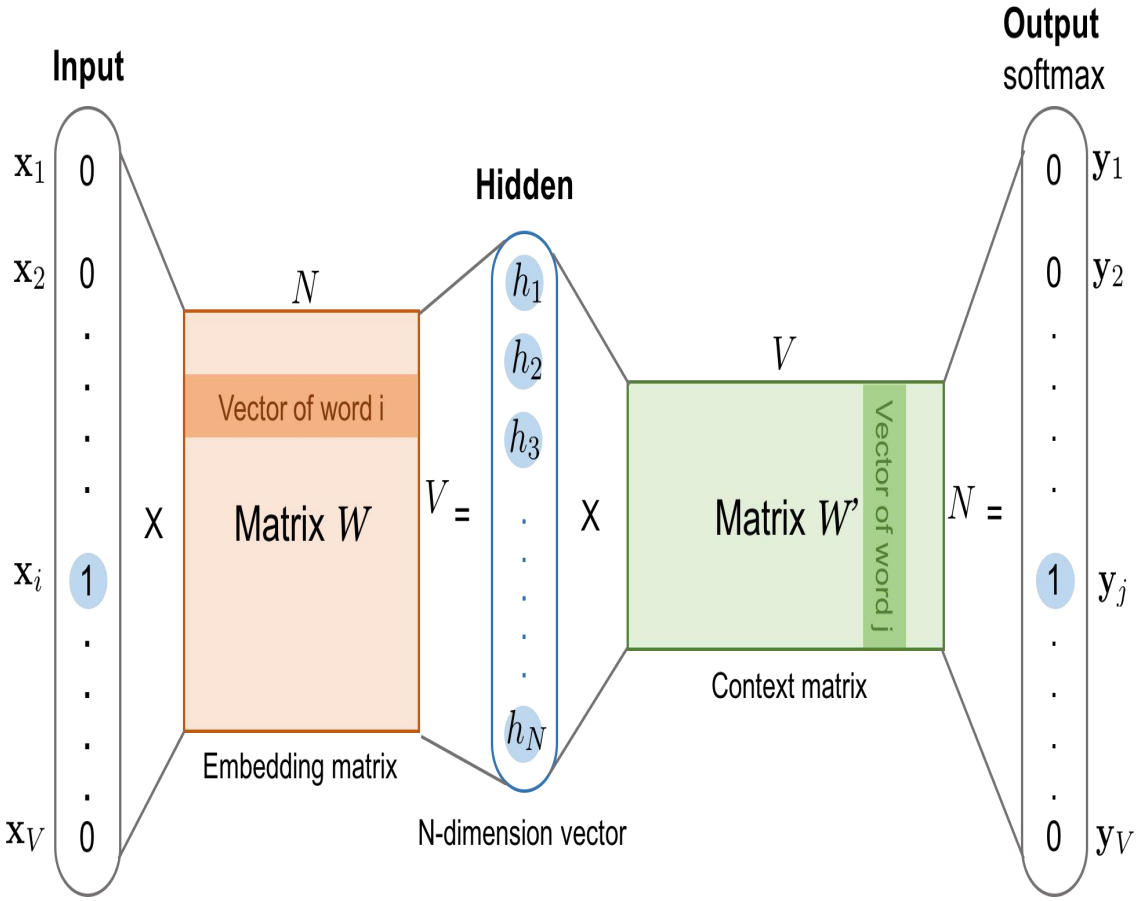
3.按规定时间在课程网站提交实验报告、代码以及PPT。

# 1、基本原理：embedding




Embedding是一个相对低维的空间，可以将高维向量转换到其中。Embedding使得机器学习更容易在大规模的输入上进行，比如表示单词的稀疏向量。理想情况下，Embedding通过将语义相似的输入紧密地放置在Embedding空间中来捕获输入的一些语义。

# 1、基本原理：




**具体的实现：**

如图1中所示，词嵌入层（Word embedding）使用二维矩阵来表示长文本。词嵌入将输入文本的每个词语通过空间映射，将独热表示（One-Hot Representation）转换成分布式表示（Distributed Representation），进而可以使用低维的词向量来表示每一个词语。经过词嵌入，每个单词具有相同长度的词向量表示。将各个词语的向量表示连起来便可以得到二维矩阵。得到词向量的方式有多种，常用的是Word2vec方法。若使用预训练好的词向量，在训练模型的时候可以选择更新或不更新词向量，分别对应嵌入层状态为Non-static和Static。

# 实验步骤

**1. 主要工具**

Python 3.6.9、PyTorch 1.4.0、numpy-1.16.6、

**2.数据集：**

（1）data: 诗词数据，将诗词中的字转化为其在字典中的序号表示。

（2）ix2word: 序号到字的映射

（3）word2ix: 字到序号的映射

# 实验步骤

## 1、导入相关包

```
import os
import sys
import json

import torch
import torch.nn as nn
from torchvision import transforms,
datasets, utils
import matplotlib.pyplot as plt
import numpy as np
import torch.optim as optim
from tqdm import tqdm
import torch.optim as optim
from mymodel import PoetryModel
```

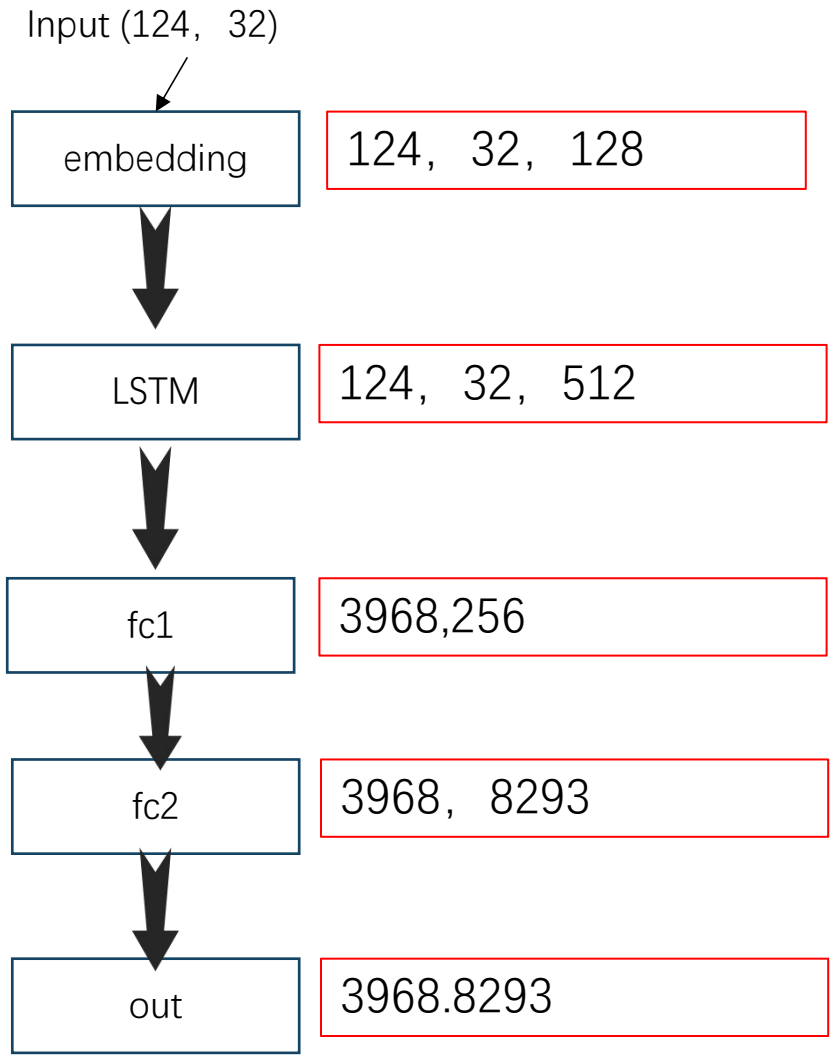
## 2、指定运算的设备

```
device = torch.device("cuda:0" if torch.cuda.is_available()
else "cpu")#判断设备是否GPU可以用，否则就指定设备
为CPU
 print("using {} device.".format(device))
```

## 3、数据的预处理：加载数据

```
# 加载我们给定的数据集
def prepareData():

    # 加载数据,其中包括三个部分：data, ix2word, word2ix
    datas = np.load("tang.npz", allow_pickle=True)
    data = datas['data']
    ix2word = datas['ix2word'].item()
    word2ix = datas['word2ix'].item()

    # 将数据转换为tensor的格式，并通过dataloader将数据加载成我们可迭代训练的数据

    data = torch.from_numpy(data)
    print(data.shape) # [57580, 125]
    dataloader = DataLoader(data,
                            batch_size = Batch_size,
                            shuffle = True,
                            num_workers = 0)
    print(len(dataloader)) # 3599

    return dataloader, ix2word, word2ix
```

## 4、构建模型




本实验设计的网络结构

```python
import torch
import torch.nn as nn
#定义我们的模型：
class PoetryModel(nn.Module):
    def __init__(self, num_embeddings, embedding_dim, hidden_dim):
        super(PoetryModel, self).__init__()
        self.embedding = nn.Embedding(num_embeddings, embedding_dim)
        self.hidden_dim = hidden_dim
        self.lstm = nn.LSTM(embedding_dim, self.hidden_dim, num_layers=2)
        self.MYliner=nn.Linear(self.hidden_dim,256)
        self.linear = nn.Linear(256, num_embeddings)

    def forward(self, input, hidden = None):
        seq_len, batch_size = input.size()

        if hidden is None:
            h_0 = input.data.new(2, batch_size, self.hidden_dim).fill_(0).float()
            c_0 = input.data.new(2, batch_size, self.hidden_dim).fill_(0).float()
        else:
            h_0, c_0 = hidden

        embeds = self.embedding(input)
        output, hidden = self.lstm(embeds, (h_0, c_0))
        output=self.MYliner(output.view(seq_len * batch_size, -1))
        output = self.linear(output)
        return output, hidden
```

## 5、模型的训练：

```
def train(dataloader, ix2word, word2ix):


    # 初始化我们的模型：
    model = PoetryModel(len(word2ix), embedding_dim, hidden_dim)

    #将模型指定到我们的设备上
    model.to(device)

    #定义优化器和损失函数
    optimizer = torch.optim.Adam(model.parameters(), lr = learning_rate)
    criterion = nn.CrossEntropyLoss()

    # 开始训练
    Loss_list=[]
    for epoch in range(epochs):
        running_loss=0.0
        for batch_idx, data in enumerate(dataloader):
            data = data.long().transpose(1, 0).contiguous()
            data = data.to(device)
            input, target = data[:-1, :], data[1:, :]
            output, _ = model(input)
            loss = criterion(output, target.view(-1))
            running_loss += loss.item()

            if (batch_idx+1) % 899 == 0 & verbose:
                print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                    epoch+1, (batch_idx+1) * Batch_size, len(dataloader.dataset),
                    100. * (batch_idx+1) / len(dataloader), loss.item()))

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
```

## 6、训练过程可视化

```
# 保存模型
torch.save(model.state_dict(), 'model.pth')
print('Finished Training')
print("开始绘制训练曲线：----------------------------")
x1 = np.arange(1, len(Loss_list) + 1)
plt.plot(x1, Loss_list, '--', color='red', label='Train_loss')
plt.title('model losses')
plt.ylabel('Loss')
plt.xlabel('epoch')
plt.legend()
plt.show()
```

## 7、predict

```
def generate(start_words, ix2word, word2ix):

    # 加载我们的模型：
    model = PoetryModel(len(word2ix), embedding_dim, hidden_dim)
    #加载训练好的模型的权重文件
    model.load_state_dict(torch.load(trained_model_path))
    #将模型指定到我们的设备中
    model.to(device)

    # 开始的句子
    results = list(start_words)
    start_word_len = len(start_words)

    input = torch.Tensor([word2ix['<START>']]).view(1, 1).long()
    input = input.to(device)
    hidden = None

    # 生成诗句小于最大的长度
    for i in range(max_gen_len):
        output, hidden = model(input, hidden)

        if i < start_word_len:
            w = results[i]
            input = input.data.new([word2ix[w]]).view(1, 1)

        else:
            top_index = output.data[0].topk(1)[1][0].item()
            w = ix2word[top_index]
            results.append(w)
            input = input.data.new([top_index]).view(1, 1)

        if w == '<EOP>':
            del results[-1]
            break

    return results
```

定义诗句生成函数，并用来测试给定的开始诗句，生成后面的诗句

## 8、藏头诗生成测试

```
def gen_acrostic(start_words, ix2word, word2ix):

    model = PoetryModel(len(word2ix), embedding_dim, hidden_dim)
    model.load_state_dict(torch.load(trained_model_path))
    model.to(device)
    results = []
    start_word_len = len(start_words)
    input = (torch.Tensor([word2ix['<START>']]).view(1, 1).long())
    input = input.to(device)
    hidden = None

    index = 0  # index of the character in start_words
    pre_word = '<START>'  # pre_word

    for i in range(max_gen_len):
        output, hidden = model(input, hidden)
        top_index = output.data[0].topk(1)[1][0].item()
        w = ix2word[top_index]
        if (pre_word in {u'。', u'！', '<START>'}):

            if index == start_word_len:
                break

            else:
                w = start_words[index]
                index += 1
                input = (input.data.new([word2ix[w]])).view(1, 1)

        else:
            input = (input.data.new([word2ix[w]])).view(1, 1)

        results.append(w)
        pre_word = w

    return results
```

定义藏头诗句生成函数，并用来测试给定的开始诗句，生成后面的诗句

# 四、实验结果



## 1、迭代十次的结果

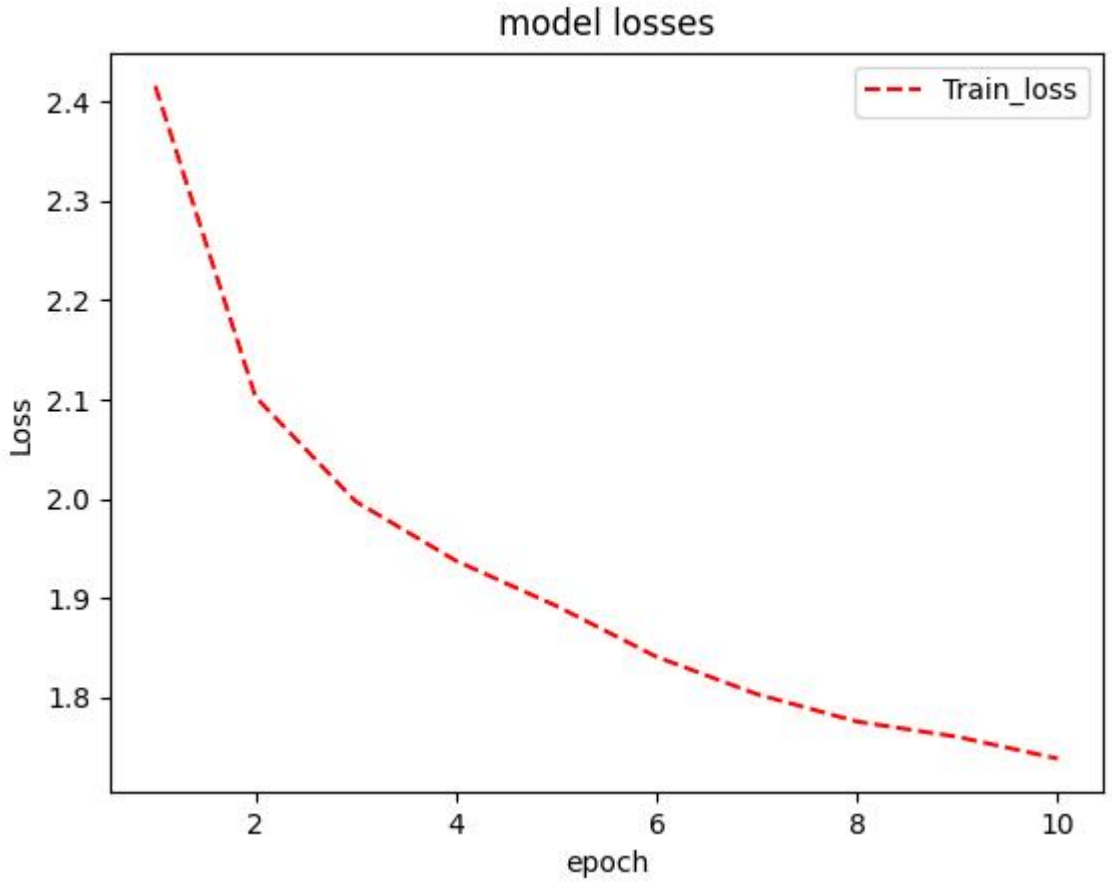
```
Train Epoch: 2 [28768/57580 (50%)]  Loss: 1.954306
Train Epoch: 2 [57536/57580 (100%)] Loss: 2.084495
Train Epoch: 3 [28768/57580 (50%)]  Loss: 2.153803
Train Epoch: 3 [57536/57580 (100%)] Loss: 2.199356
Train Epoch: 4 [28768/57580 (50%)]  Loss: 2.003785
Train Epoch: 4 [57536/57580 (100%)] Loss: 2.106595
Train Epoch: 5 [28768/57580 (50%)]  Loss: 1.655022
Train Epoch: 5 [57536/57580 (100%)] Loss: 2.062831
Train Epoch: 6 [28768/57580 (50%)]  Loss: 1.789921
Train Epoch: 6 [57536/57580 (100%)] Loss: 2.177508
Train Epoch: 7 [28768/57580 (50%)]  Loss: 1.806935
Train Epoch: 7 [57536/57580 (100%)] Loss: 1.943591
Train Epoch: 8 [28768/57580 (50%)]  Loss: 1.941552
Train Epoch: 8 [57536/57580 (100%)] Loss: 1.825491
Train Epoch: 9 [28768/57580 (50%)]  Loss: 1.800300
Train Epoch: 9 [57536/57580 (100%)] Loss: 1.587843
Train Epoch: 10 [28768/57580 (50%)] Loss: 1.887935
Train Epoch: 10 [57536/57580 (100%)]    Loss: 1.587747
Finished Training
```

## 2、最后一次的损失和准确率

[epoch 30]:  train_loss: 1.5877

**迭代10次之后，训练集损失达到1.5877**

## 2、迭代十次损失变化曲线



**3、诗句生成结果：**

'湖', '光', '秋', '月', '两', '相', '和',
'江', '上', '江', '南', '雨', '气', '多', '。',
'江', '上', '有', '时', '看', '月', '好', '，',
'江', '南', '有', '客', '见', '君', '歌', '。',
'江', '南', '水', '阔', '无', '人', '到', '，',
'江', '上', '江', '南', '有', '客', '过', '。',
'江', '上', '有', '时', '应', '有', '路', '，',
'江', '南', '有', '树', '不', '无', '多', '。',
'江', '南', '江', '上', '无', '人', '钓', '，',
'江', '上', '江', '村', '有', '钓', '歌', '。',
'岸', '上', '鸬', '鹚', '千', '里', '暮', '，',
'江', '头', '鹭', '鹡', '一', '声', '歌', '。',
'岸', '花', '落', '尽', '千', '樯', '下', '，',
'江', '水', '相', '通', '万', '里', '波', '。',
'岸', '上', '鸬', '鹚', '惊', '鸟', '语', '，',
'岸', '边', '红', '叶', '落', '渔', '歌', '。'

**4、藏头诗生成：**

'轻', '舟', '山', '上', '木', '，', '高', '下', '无', '人', '家', '。',
'舟', '中', '有', '一', '径', '，', '石', '上', '有', '石', '泉', '。',
'已', '有', '漙', '阳', '亭', '，', '其', '气', '无', '所', '鲜', '。',
'过', '时', '不', '可', '见', '，', '日', '暮', '空', '林', '间', '。',
'万', '里', '无', '人', '事', '，', '一', '言', '无', '所', '牵', '。',
'重', '门', '有', '余', '悲', '，', '一', '榻', '无', '所', '缘', '。',
'山', '中', '有', '老', '翁', '，', '不', '觉', '老', '病', '年', '。'

# 请老师批评指正！

THANK YOU